**Atharv R. Aundhkar roll no : 4**

**Practical no 1**

**Aim =** Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

**Code = DFS**

def dfs(graph, start\_node, goal\_node):

visited = set()

stack = [(start\_node, [start\_node])]

while stack:

(current\_node, path) = stack.pop()

if current\_node == goal\_node:

return path

visited.add(current\_node)

for neighbor in graph[current\_node]:

if neighbor not in visited:

stack.append((neighbor, path + [neighbor]))

return None

graph = {

'A': ['B', 'C'],

'B': ['A', 'D'],

'C': ['A', 'E'],

'D': ['B', 'E', 'F'],

'E': ['C', 'D', 'F'],

'F': ['D', 'E']

}

start\_node = 'B'

goal\_node = 'D'

path = dfs(graph, start\_node, goal\_node)

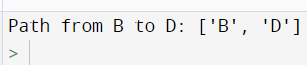
if path is not None:

print(f"Path from {start\_node} to {goal\_node}: {path}")

else:

print(f"No path found from {start\_node} to {goal\_node}")

**Graph =**

**Output =**

**BFS**

**Code** =

import collections

def bfs(graph, root,goal):

visited, queue = set(), collections.deque([root])

visited.add(root)

while queue:

vertex = queue.popleft()

for current\_node in graph[vertex]:

if current\_node not in visited:

visited.add(current\_node)

queue.append(current\_node)

if goal==current\_node:

print(visited)

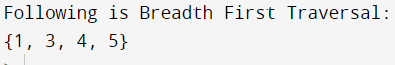
graph = {1: [3, 4], 3: [5], 5: [6], 4: [7], 7: [9, 10], 6: [9], 9: [6], 10: [7]}

print("Following is Breadth First Traversal: ")

bfs(graph,1,5)

**Graph =**

**Output =**

****

Shravani D. Kadam Roll no = 23

**Practical no 3**

**Aim :** Implement Greedy search algorithm for Selection Sort

**Code :**

def Selection\_Sort(array):

for i in range(0, len(array) - 1):

smallest = i

for j in range(i + 1, len(array)):

if array[j] < array[smallest]:

smallest = j

array[i], array[smallest] = array[smallest], array[i]

array = input('Enter the list of numbers: ').split()

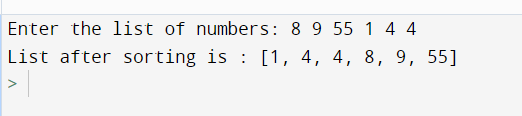
array = [int(x) for x in array]

Selection\_Sort(array)

print('List after sorting is : ', end='')

print(array)

**output :**



**Atharv R. Aundhkar Roll no : 4**

**Practical no 2**

**Aim =** Implement A star Algorithm for any game search problem.

**Code =** import heapq

# Define the goal state

goal\_state = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

# Define the heuristic function h(state)

def h(state):

return sum(abs(state[i][j]//3 - i) + abs(state[i][j]%3 - j) for i in range(3) for j in range(3) if state[i][j])

# Define the A\* search function

def a\_star(start\_state):

heap = [(h(start\_state), start\_state, 0)]

visited = set()

while heap:

(cost, state, g) = heapq.heappop(heap)

if state == goal\_state:

return g

if str(state) in visited:

continue

visited.add(str(state))

for (i, j) in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

new\_state = [row[:] for row in state]

row, col = find\_zero(new\_state)

new\_row, new\_col = row+i, col+j

if 0 <= new\_row < 3 and 0 <= new\_col < 3:

new\_state[row][col], new\_state[new\_row][new\_col] = new\_state[new\_row][new\_col], new\_state[row][col]

heapq.heappush(heap, (g+h(new\_state), new\_state, g+1))

return -1

# Define a function to find the location of the empty cell (0)

def find\_zero(state):

for i in range(3):

for j in range(3):

if state[i][j] == 0:

return i, j

# Define a function to print the state

def print\_state(state):

for i in range(3):

for j in range(3):

print(state[i][j], end=' ')

print()

# Define the start state

start\_state = [[0, 2, 3],

[1, 4, 6],

[7, 5, 8]]

# Print the start state

print("Start state:")

print\_state(start\_state)

# Print the goal state

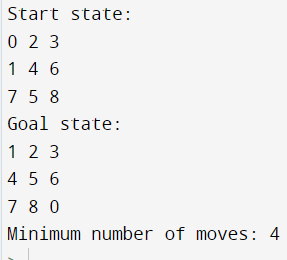
print("Goal state:")

print\_state(goal\_state)

# Compute the minimum number of moves required to reach the goal state from the initial state

cost = a\_star(start\_state)

print("Minimum number of moves:",cost)

**output =**

**Atharv R. Aundhkar Roll no : 4**

**Practical no**

**Aim** = Develop an elementary chatbot for any suitable customer interaction application.

**Code** = import random

# Define some responses

responses = {

"hi": ["Hello! , how can i help you..??"],

"how are you": ["I'm doing well, thanks for asking.", "I'm fine, how about you?", "Not bad, and you?"],

"goodbye": [ "Thankyou,hope we could help you out"],

"default": ["Sorry, I don't understand.", "Could you please rephrase that?", "I'm not sure what you mean."],

"what is this product?":["This is iphone 11."],

"variant":["Ram : 64GB & processor : A13 Bionic "],

"specification":["Brand : Apple ,IP rating : IP68 ,Display: 6.1-inch (15.5 cm diagonal) Liquid Retina HD LCD display "],

"price":["40,999 /-"],

"colours available":["Black,Gold,Blue"],

"camera":["12MP TrueDepth front camera"],

"o.s":["iOS 14"],

"costumercare":["9874563211 or iphone@gmail.in"]

}

# Define the chatbot function

def chatbot():

# Print a welcome message

print("Welcome to the chatbot!")

print("Type 'goodbye' to exit.\n")

# Start the conversation

while True:

# Get the user's input

user\_input = input("You: ")

# Check if the user wants to exit

if user\_input.lower() == "goodbye":

print(random.choice(responses["goodbye"]))

break

# Look for a response in the responses dictionary

response = responses.get(user\_input.lower(), random.choice(responses["default"]))

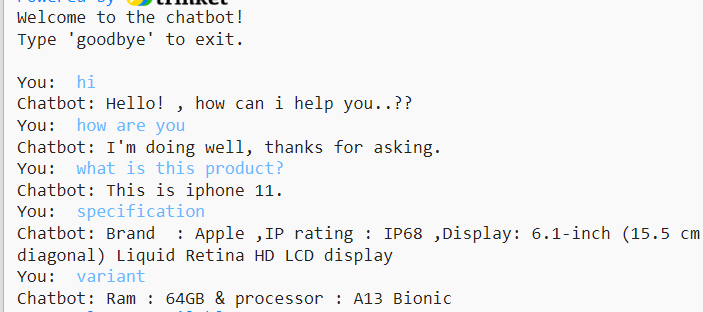
# Print the chatbot's response

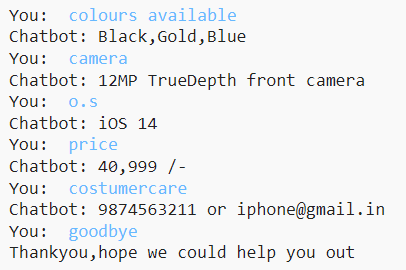
print("Chatbot:" , random.choice(response))

# Call the chatbot function

chatbot()

**Output =**





**Atharv R. Aundhkar Roll no : 4**

**Practical no**

**Aim =** Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

**Code =**

def n\_queen(n):

# Create an empty chessboard

board = [[0 for x in range(n)] for y in range(n)]

def is\_safe(row, col):

# Check if there is a queen in the same row

for i in range(col):

if board[row][i] == 1:

return False

# Check if there is a queen in the upper diagonal on the left side

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

# Check if there is a queen in the lower diagonal on the left side

for i, j in zip(range(row, n, 1), range(col, -1, -1)):

if board[i][j] == 1:

return False

# If all conditions are satisfied, then the position is safe

return True

def solve(col):

# If all queens are placed, then return True

if col >= n:

return True

# Try placing a queen in each row of the current column

for row in range(n):

if is\_safe(row, col):

# Place the queen on the board

board[row][col] = 1

# Recursively solve for the remaining columns

if solve(col + 1):

return True

# If placing the queen in the current row and column doesn't lead to a solution,

# then remove the queen from the board and try the next row

board[row][col] = 0

# If no queen can be placed in the current column, then return False

return False

# Start solving the problem from the first column

if solve(0):

# Print the solution if it exists

for i in range(n):

for j in range(n):

print(board[i][j], end=' ')

print()

else:

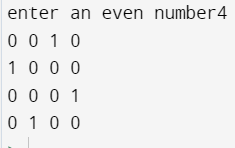
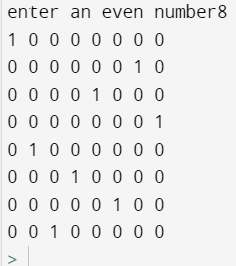
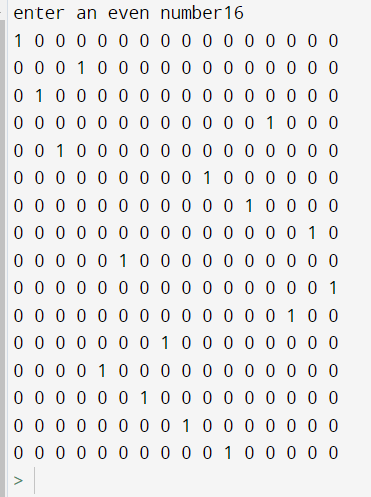
# If no solution exists, then print an error message

print("No solution exists.")

x=int(input("enter an even number"));

n\_queen(x)

**output=**



\

Practicl6

**Aim =**  **Implementinng Expert System for Hospital and Medical facilities**

**Code =**

def get\_user\_input(prompt):

while True:

try:

return float(input(prompt))

except ValueError:

print("Please enter a valid number.")

def diagnose\_disease():

print("Welcome! This expert system will help you distinguish between diseases with similar symptoms.")

age = get\_user\_input("What is the patient's age? ")

body\_temp = get\_user\_input("What is the patient's body temperature? ")

oxy\_level = get\_user\_input("What is the oxygen level? ")

symptoms = [

('cough and sore throat', ['Flu', 'Common Cold']),

('runny nose', ['Flu', 'Common Cold']),

('sneezing', ['Common Cold', 'COVID-19']),

('headache', ['Flu']),

('body/muscular aches', ['Flu', 'COVID-19']),

('regular tiredness', ['Flu', 'COVID-19', 'Pneumonia']),

('fever', ['Flu', 'Common Cold', 'COVID-19']),

('vomiting or diarrhea', []),

('shortness of breath and chest pain', ['COVID-19', 'Pneumonia']),

('lost your sense of smell or taste', ['COVID-19'])

]

disease\_counts = {'Flu': 0, 'Common Cold': 0, 'COVID-19': 0, 'Pneumonia': 0}

for symptom, diseases in symptoms:

answer = input(f'Are you experiencing {symptom}? (Y/N) ').lower()

if answer == 'y':

for disease in diseases:

disease\_counts[disease] += 1

if all(count == 0 for count in disease\_counts.values()):

print('Congratulations! You are healthy!')

else:

disease = max(disease\_counts, key=disease\_counts.get)

print(f'Based on the symptoms, you may have {disease}.')

if \_\_name\_\_ == "\_\_main\_\_":

diagnose\_disease()